CS172 Computer Vision I:
Homework 2:Implementation of Geometry to the rescue

Ju Yiang
Student ID 2021533088
juya@shanghaitech.edu.cn

Abstract

In this assignment, my work are as follows:
1. downloading and formatting KITTI dataset
2. solving environment conflicts of monodepth2 and use it train an unsupervised CNN model on KITTI dataset
3. Experiment on methods solving occlusion problem, and methods using geometry information to improve performance.

1. Preparation works

1.1. Dataset

KITTI raw dataset is very large(about 175GB), in this experiment, I follow the settings of Geometry to the rescue, download the categories of "city" and "roads", which sums up to about 18GB instead. For train and value splitting, I use the "eigen-full" split in monodepth2. Since my dataset is a subset of KITTI, I use a python script to remove the items in "eigen-full" that are not in my dataset, eventually splits my dataset into 7118 training items and 274 validation items.

1.2. Environment conflicts

The GPU on my computer belongs to RTX40 series, its lowest supported CUDA version is 11.8, so I cannot use the original environment that monodepth2 requires, this leads to a lot of conflicts caused by torch and python version. I spent a few hours solving these conflicts.

2. Method

2.1. Dataloader and augmentation

Each training batch is a list of images, either left angle of right angle of the stereo pairs, the dataloader would do the following work to each training image:
1. load this image
2. load its opposite image(e.g. if this image is a left angle, load the correspondent right angle, vice versa.)
3. downsample left image and right image to several scales
4. employ color augmentation with a probability of 50%
5. extract depth map from velodyne points(not used in training process)

2.2. Network

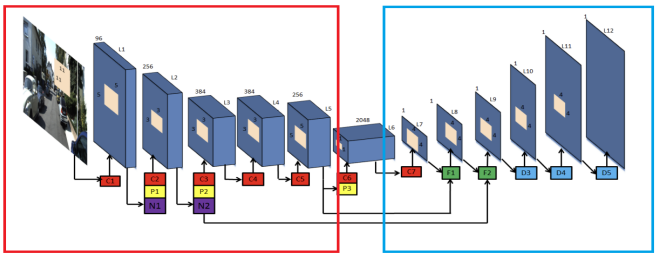


Figure 1. Network Structure

As depicted in Figure 1., the network follows basic Structure of Geometry to the Rescue, composed of an encoder(Red) and decoder(Blue). Note that in my implementation, the decoder is replaced with a resnet18 network, while the decoder is composed of several upconv layers.

2.3. Loss function

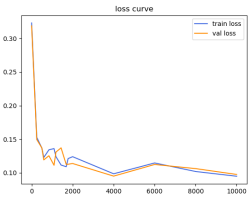
The loss function is the core of this method, instead of using the generated depth map as ground truth, it first generated a warp image using the correspondent angle, then calculated its difference with the original image.

$$Training\ loss = L_P = \sum_{t'} pe(I_t, I_{t'\to t})$$

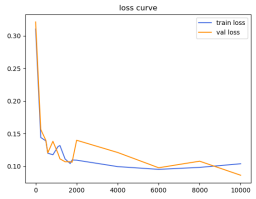
photometric reconstruction error:

$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - SSIM(I_a, I_b)) + (1-\alpha)\|I_a - I_b\|_1$$

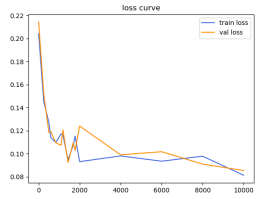
where I_a is the original image, I_b is the warped image, α is a weight parameter



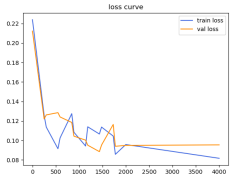
(a) simple method



(b) simple + min



(c) simple + min + automasking



(d) simple + min + automasking + flipping

2.4. Train settings

I train my network on a RTX4070 laptop GPU, it took about 2 hours to finish training.

- Environment I used is python==3.8.18, CUDA == 11.8, torch == 2.1.0, torchvision == 0.16.0, tensorboardX == 1.4.
- Hyper-parameters are as follows:
  learning rate = 1e-4
  batch size = 12
  number of epochs = 20
  scheduler step size = 15

3. Experiments

In this part I will describe my experiments, each method will have two sub-section, the first one describe its method, the second one analyze its actual performance.

3.1. simple method

In this part I employs the simple method, using only photometric reconstruction error. Loss curve is shown in Figure(a)

3.2. effect of min

In this part I compare the result using min and not using min. Loss curve is shown in Figure(b)

3.2.1 min method

Full name of min method is Per-Pixel Minimum Reprojection Loss [1], it reduce the negative effect of occlusion by using a minimum photometric error:

$$Training\ loss = L_p = \min_{t'} pe(I_t, I_{t'\to t})$$

3.2.2 performance of min

As shown in the Table 1., min helps improve the evaluation performance a little bit.

3.3. effect of automasking

In this part I compare the result using automasking and not using automasking. Loss curve is shown in Figure(c)

3.3.1 Automasking method

Automasking [1] deal with the problem of "holes" on depth map caused by moving objects, it uses a mask to ignore these objects. Formula is given as follows:

$$\mu = [\min_{t'} pe(I_t, I_{t'\to t}) < \min_{t'} pe(I_t, I_{t'})]$$

$$L_s = |\partial_x d_t^*| e^{-|\partial x I_t|} + |\partial_y d_t^*| e^{-|\partial y I_t|}$$

$$New\ Training\ loss = \mu L_p + \lambda L_s$$

3.3.2 performance of automasking

As shown in the Table 1., automasking did improve the evaluation performance.

3.4. effect of flipping

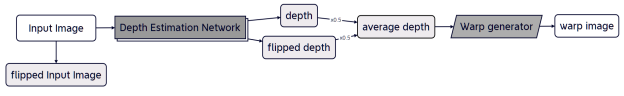


Figure 2. flipping method

3.4.1 Flipping method

Flipping method is mentioned in Forget About the LiDAR(Gonzalez et al.) [2] and PlaneDepth(Ruoyu Wang et al.) [3], I created an naive version of the method, its main concept is training with only left or right image(same with original method), for each input image, import both itself and its flipped image into the network to obtain a depth map and a flipped depth map, then generate a warp image and a flipped warp image, then flip back the flipped warp image and do average process with the warp image, the final out put warp image would suffer less occlusion effect.(see figure 2)
Reanson why this method may work is that since we train with only left or right image, the network would only have occlusion on one side. Then we generate a flipped warp image, the occlusion effect will happens on the other side, so we can average out the effect.
This experiment is splitted into two phase, in the first phase, I train with left angles only, in the second phase, I train with right angles only.

3.4.2 performance of flipping

My naive version of flipping method did not improve the evaluation performance, training result after the first phase is disappointing, so I added the second phase, it only improved a little bit after train on right angles.

Version	**abs-rel**	**sq-rel**	**rmse**	**rmse-log**
simple model	0.137	1.383	5.767	0.225
min	0.135	1.431	5.763	0.224
automasking	0.137	1.353	5.708	0.229
flipping1	0.181	2.584	8.202	0.276
flipping2	0.178	2.433	8.017	0.276

Table 1. Evaluation metrics

References

[1] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation, 2019.

[2] Juan Luis Gonzalez and Munchurl Kim. Forget about the lidar: Self-supervised depth estimators with med probability volumes, 2020.

[3] Ruoyu Wang, Zehao Yu, and Shenghua Gao. Planedepth: Self-supervised depth estimation via orthogonal planes, 2023.